DOCUMENT RESUME

ED 097 899 IR 001 267

AUTHOR Peelle, Howard A.
TITLE The Computer "Glass Box": Teaching With A Programming Language.
INSTITUTION Massachusetts Univ., Amherst. School of Education.
PUB DATE May 74
NOTE 25p.; Paper presented at the Shared Educational Computer Systems (SECOS) Conference (New Paltz, New York, May 1974)

EDRS PRICE MF-$0.75 HC-$1.85 PLUS POSTAGE
DESCRIPTORS Computer Assisted Instruction; *Computer Programs; *Concept Formation; Learning Processes; Programing Languages
IDENTIFIERS APL; A Programing Language; Computer Glass Box

ABSTRACT

Using A Programing Language (APL), a "computer glass box" was designed to stimulate students to think about selected concepts as well as to elucidate and reveal understanding. This approach is pedagogically suitable for a wide range of educational levels--from elementary school children to university graduate students. Using APL computer programs, students can proceed to learn during several complementary activities. Specifically, they can: examine, analyze, predict, execute, scrutinize, experiment, modify, generalize, invent, and discuss. The ideal APL is also expository; it "speaks" to its reader, explicating concepts and procedures in concrete terms. (WCM)

THE COMPUTER "GLASS BOX":

TEACHING

WITH

A PROGRAMMING LANGUAGE

By

Howard A. Peelle

University of Massachusetts

May 1974

THE COMPUTER "GLASS BOX":

TEACHING

WITH

A PROGRAMMING LANGUAGE

Howard A. Peelle
University of Massachusetts

Introduction

The COMPUTER GLASS BOX is a bold new approach to teaching with A Programming Language.[1] In this approach, short and quickly comprehensible computer programs are given to students for their direct viewing. Each program embodies a concept, a procedure, or a relationship and is written as simply and clearly as possible. The inner workings of such a program are visible and, hence, become the basis for learning.

This approach utilizes a computer program more as a "glass box" than a black box. The program's formal definition -- expressed in the explicit terms of a programming language -- serves to elucidate and reveal understanding. By observing the structure of a program as well as its behavior, key concepts may become transparent to the student.

---

[1] A Programming Language (abbreviated APL) is a new multi-purpose computer programming language developed by Kenneth Iverson of IBM. Originally conceived as a unifying mathematical notation, APL has since been used successfully in fields such as business, scientific research and education.

Related Research

The glass box approach represents a synthesis of ideas put forth by three other researchers. MIT's Seymour Papert has recommended that children study procedures actively by using a computer programming language (called LOGO) as a conceptual framework [1]. Kenneth Iverson of IBM has persistently stressed simplicity and generality in using APL to expose fundamentals in a variety of mathematical and scientific disciplines [2]. IBM's Paul Berry first advocated open use of APL as a strategy for teaching in what he called the "functional approach" [3].

Characteristics of the COMPUTER GLASS BOX Approach

In contrast to conventional computer-assisted instruction (CAI), the glass box approach allows the student significant control over his own learning processes. This control is achieved through the activity of programming. Programs can be entered independently by the student via a computer terminal, and their use requires no other pre-stored curriculum material--as do most CAI applications. Indeed, making the full power of the computer accessible to the learner is 180° from the kind of CAI characterized by programmed instruction, tutorial, or drill-and-test sequences.

This approach is pedagogically suitable for a wide range of educational levels--from elementary school children to university graduate students. Especially for children who have been held powerless in lock-step educational systems, use of the computer in this way opens up new worlds of learning--active learning, learning with power.

Using glass box computer programs, students can proceed to learn during several complementary activities. Specifically, they can:

examine the program's definition (intuitively)

analyze the program's definition (logically)

predict the outcomes of the program

execute the program on a computer

scrutinize the program's behavior

experiment with different applications of the progra

modify or expand the program

generalize the program

invent new or related programs, and

discuss implications with teachers and peers.

These student-initiated, student-responsible, success-oriented activities differ dramatically from frantic hand-waving about abstract concepts often seen in classrooms.

The ideal glass box program is also expository--it 'speaks' to its reader, explicating concepts and procedures in concrete terms. Desirable characteristics of such a program are:

Simplicity

Comprehensibility

Flexibility

Generality

Elegance

Provocative Implications

By "simplicity" I mean that a single idea of modest scope is to be taught using a brief program (about 10 lines of APL coding, taking less than 5 minutes to type). By "comprehensibility", I mean using clear, readable commands (usually one per line) with well-chosen mnemonic identifiers. By "flexibility" I mean a program design which is easily modified and which can be used with other programs in modular structuring (nested sub-programs with explicit resultants). By "generality" I mean developing mathematical models which can extend to a class of cases. By "elegance" I mean choosing expressions which strike one's aesthetic chords. And, finally, a glass box program is "provocative" when its implications suggest interesting follow-up discussions.

To the extent that these characteristics foster insight and learning, a glass box program is, itself, a pedagogical agent.

Examples of Glass Box Programs

To illustrate this approach, some sample glass box APL programs are described below, with accompanying suggestions for extending their use in teaching-learning settings. The sample programs are chosen from special topics in the following areas:

Computer Assisted Instruction

Psychology

Cybernetics

Computer Art

COMPUTER - ASSISTED INSTRUCTION

In order to emphasize the contrast with conventional uses of computers for teaching, the first glass box program illustrated is from the area of computer-assisted instruction. Instead of concealing the CAI program -- usually designed to control the child's behavior -- we show him the mechanism itself so that he may see how it works and ultimately control the computer.

Consider the APL program below which exposes the essence of drill-and-practice in multiplication skills. In drill-and-practice, typically, a student is given a series of problems to solve, is asked for his answers, and the answers are judged for correctness, etc. Indeed, the computer is an excellent vehicle for administering drill-and-practice, but a programming language can also describe this process clearly.

```
∇ DRILL
[1]   NEWPROBLEM:
[2]   'MULTIPLY'
[3]   □←FIRST←?20
[4]   □←SECOND←?20
[5]   ENTER:ANSWER←□
[6]   →NEWPROBLEM IF ANSWER=FIRST×SECOND
[7]   'NOPE. TRY AGAIN.'
[8]   →ENTER
∇
```

The DRILL program begins with a NEWPROBLEM and prints 'MULTIPLY', a simplified message telling the student what to do with the two numbers that will follow. The FIRST number is an integer randomly chosen between 1 and 20, and the SECOND number likewise.

The student may ENTER his ANSWER which is then judged for correctness by the program. IF the ANSWER equals the FIRST number times the SECOND number, a NEWPROBLEM is given; otherwise (if ANSWER is wrong) 'NOPE. TRY AGAIN.' is printed, and the student may ENTER his answer again.

NOTE: IF is a sub-program used to facilitate the reading of branching commands. Its definition is:

```
      ∇ BRANCH←LINE IF CONDITION
[1]     BRANCH←CONDITION/LINE
      ∇
```

Its syntax is → (line number) IF (condition)

It means that IF the condition is true (evaluates to 1), the program branches to the line number (or line label) given; IF the condition is false (evaluates to 0), the program branches to the next line.

In order to use the DRILL program, its name is typed. The following is a sample:

```
                    DRILL

          MULTIPLY

           19

           2

          ?:

                38

          MULTIPLY

           16

           18

          ?:

                248

          NOPE. TRY AGAIN.

          ?:

                288

          MULTIPLY

           8

           12

          ?:

                96

          MULTIPLY

           6

           2

          ?:

                12

          MULTIPLY

           14

           18

          ?:
```

Students notice immediately that this program has a flaw. It does not stop! Scrutinizing the program's definition reveals that after getting a multiplication problem correct, one always gets a new problem -- ad infinitum. Also, after getting a problem wrong, the student must answer that same problem again -- another potentially endless loop. The student's first task, then, might be to build in an option to stop the program at will.

DRILL is, of course, only a <u>prototype</u> program. With other modifications of one's choosing, DRILL may become considerably more sophisticated. Possible extensions include: (a) displaying pictorial feedback -- like a "smiley face" for positive reinforcement

```
  *****
 * o o *
*  V  *
* \_/ *
 *****
```

or a "grouchy face"

```
  *****
 * * * *
*  ^  *
* /_\ *
 *****
```

instead of 'NOPE. TRY AGAIN.', (b) presenting a pre-specified total number of problems, (c) limiting the number of allowable mistakes on individual problems (or all problems), (d) generalizing the multiplicands to create a more flexible range of problems (including negative numbers, decimals, etc.), (e) gathering performance data, (f) using performance criteria to make diagnoses, (g) automatically adapting level of difficulty based on diagnoses, (h) adding personalized instructions, and (i) building in timing components, jump-ahead options and hints.

PSYCHOLOGY

With computer programs suitable for viewing, students may learn some fundamentals of psychology. In studying behavior, for example, consider the following APL program[1] which models -- albeit crudely -- an emotional reaction.

```
     ∇ TEMPER
[1]    EMOTION←0
[2]    NEW:EMOTION←⌷+EMOTION÷2
[3]    →MAD IF EMOTION>10
[4]    →NEW
[5]    MAD:'**!?!**?!!'
     ∇
```

TEMPER is a program which will, under certain conditions, 'get mad at you'.

The program begins with zero EMOTION and then encounters a series of numbers, representing 'events' in the life of the program. A low number is low in emotional significance; whereas high numbers are highly emotion-producing.

Each time a number is entered, the program generates a NEW EMOTION based on a simple mathematical model: EMOTION becomes the number just entered plus one half of the previous EMOTION. (In the course of human events, this might be akin to the ameliorating effect of time on emotional burdens, i.e. 'sleeping on your troubles'.)

This process continues until a test condition--the "threshold" for mad behavior--is exceeded. The program goes MAD if EMOTION ever becomes greater than 10. (**!?!**?!! is the computer's programmed vernacular.)

---

[1]This program is similar to one written in a simplified FORTRAN by John Loehlin in Computer Models of Personality, Random House, NY, 1968.

To use the program, the child types its name (TEMPER) and then enters a sequence of numbers. For example:

```
                                        TEMPER
                                  □:
Where a 4 is like "stubbing
your toe",                                4
                                  □:
6 is like "losing your
wallet," and                              6
                                  □:
8 is like "missing the
last bus."                                8
```

This sequence produced MAD behavior. **!?!**?!!

But, suppose one tries entering the same numbers in a different order:

```
TEMPER

□:

   8

□:

   6

□:

   4

□:

→
```

Here, the program does not display MAD: '**!?!**?!!'. Apparently, (for this model) the sequence 8 6 4 is "tolerable," whereas the previous sequence 4 6 8 clearly was not tolerable!

Again, this suggests an analogy with human behavior: experiencing the most emotion-packed events first and then tapering off may be more tolerable than the reverse.

Other variations of input also suggest interpretation in terms of human psychology. Sandwiching a low-emotion event between two high-emotion events, say 7 2 7, can make the total sequence tolerable; by contrast, the events 7 7 2 and 2 7 7 produce mad behavior.

The mathematics underlying this TEMPER model can be exposed quickly and naturally. For example, after some experimentation with the program, one might wonder: How many 5s can the program take before it 'blows its top?'

TEMPER

□:

5

□:

5

□:

5

□:

5

□:

5

□:

A sequence of 5s builds up EMOTION to higher and higher values, but never reaches 10. This process parallels the well-known geometric series $1 \quad \frac{1}{2} \quad \frac{1}{4} \quad \frac{1}{8} \quad \frac{1}{16} \quad \frac{1}{32} \cdots$ the sum of which converges to 2. Exploring in this way, a child may gain some insight into the nature of infinite series in an active and interesting (at least less abstract) setting.

Some simple modifications of the TEMPER program students might make are to: (a) change the threshold, e.g. from 10 to 25 for higher tolerance, or to ?25 (a random number) for unpredictable behavior; (b) modify the model, e.g. from EMOTION ÷ 2 to EMOTION ÷ 3 to express stronger 'forgetting'; (c) adapt the program for use by others, e.g. inserting conversational statements such as 'ENTER NUMBERS FROM 1 TO 9' or even 'CAUTION! THIS PROGRAM MAY BECOME EMOTIONAL...', and (d) make the program dynamic, e.g. automatically resetting EMOTION to 0 after an emotional catharthis.

Possible extensions of TEMPER include: (a) writing related programs, such as a version with multiple emotional dimensions like ANGER, FEAR, and LOVE, and (b) writing companion programs, such as two TEMPER-like programs which interact with each other so that one's output is the other's input.

CYBERNETICS

In the area of cybernetics, students can be introduced to some sophisticated ideas by using simple computer programs. Scene analysis, for example, is an important part of robotics research. In designing vision machines, it is important to know what types of scenes can be computationally distinguished. Consider the two scenes below:

SCENES

One scene is "connected"; the other is not connected. Note that the same line segments comprise the two scenes, but that they are in different positions.

Suppose one of these two SCENES is PICKed at random. Call it MYSTERY.

MYSTERY ← PICK SCENES

Further suppose that you are permitted to PEEK at small portions of the MYSTERY scene -- called "microscenes" -- but you are not told <u>where</u> the microscenes came from. For example,

PEEK MYSTERY

```
****
*
*
```

PEEK MYSTERY

```
****
```

PEEK MYSTERY

```
*
***
```

PEEKing at MYSTERY is like using a flashlight to illuminate small unidentifiable places on a much larger unknown scene.

After a period of probing, the question arises: <u>Can</u> you determine which scene it is that you are looking at?[1] (The answer is postponed so that the reader may ponder this question.)

---

[1]This question is treated as a theorem by Minsky and Papert in their book <u>Perceptrons</u>, MIT Press, 1970.

The APL programs which facilitate exploration of this question in scene analysis are simple indeed:

```
      ∇ MYSTERY←PICK SCENES
[1]     MYSTERY←SCENES[?2;;]
      ∇
```

This program will PICK one of 2 SCENES at random for the result called MYSTERY.

```
    ∇ MICROSCENE←PEEK SCENE
[1]   MICROSCENE←SCENE[(⍳4)+(?10);(⍳4)+(?10)]
    ∇
```

This program will PEEK at some two-dimensional SCENE and produce a random 4 by 4 portion for the result called MICROSCENE.

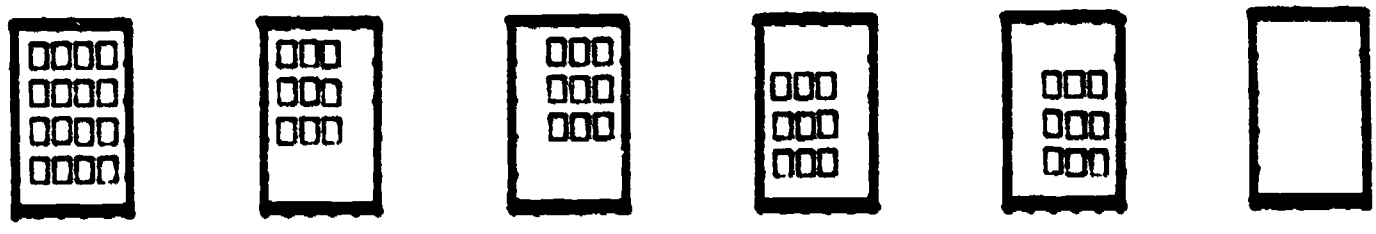
The enterprising student might elect to automate the production of MICROSCENEs.

```
      AUTOPEEK
```


```
      ∇ AUTOPEEK
[1]     ''
[2]     PEEK MYSTERY
[3]     ''
[4]     →1
      ∇
```

Soon it should become clear that these two SCENES cannot be distinguished on the basis of random microscenes alone. (Of course, if one could trace sequentially through a scene, its "connectedness" or "non-connectedness" could be determined easily.)

Possible extensions of this excursion into scene analysis include studying perceptrons[1] and related questions about "spatially local evidence." For example, if all the possible microscenes look like these:



(plus geometric translations of these)

Can you determine the type of scene from which they were drawn?

(This one is left for the reader.)

---

[1]Perceptrons are theoretical machines which can be trained to detect features of a scene by computations in a layered network of logical elements.

COMPUTER ART

The world of computer art can be opened to students through a few simple APL programs. Beginning with a foray into automated design, they can proceed to engage matters of aesthetic judgement and artistic technique.

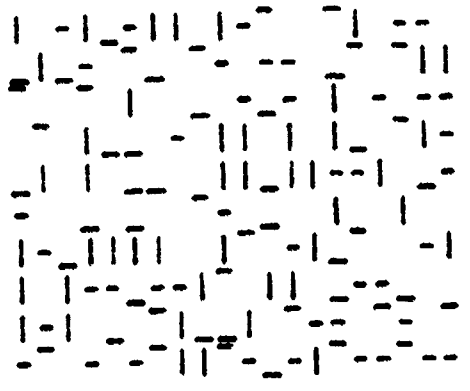
For example, consider the following DESIGN program.

```
     ∇ PICTURE←SIZE DESIGN COLORS
[1]    HOWMANY←⍴COLORS
[2]    PICTURE←COLORS[?SIZE⍴HOWMANY]
     ∇
```

DESIGN uses some COLORS (symbols on the keyboard) and some SIZE (two dimensions of a matrix) to produce a PICTURE.

A simple program like DESIGN goes a long way with children. They seem never to tire of it, for it can produce quite a variety of designs:

```
      10  20  DESIGN  ' ¯-||-_ '
```



```
10  20  DESIGN  '//Δ●▽\\'
```

```
Δ/Δ▽//Δ▽\▽\▽\▽\\●/\\
▽/●▽▽Δ▽/▽\/Δ\///Δ/\▽
\▽▽///▽●/\●\\/\\●\▽\
▽/▽●/\Δ//▽\//\//▽Δ▽▽
Δ▽\\●▽Δ\▽\/▽/Δ▽/\\\\
/●●//Δ▽●///\Δ/●/▽Δ●●
//Δ\▽/\▽/▽\/\●●Δ/▽\\
▽▽▽\\●\●▽/\Δ▽Δ●ΔΔΔΔ/
\/●\▽▽▽Δ\●▽\//\●\▽Δ\
▽●Δ\\\▽▽\Δ●▽\Δ\\\Δ\Δ
```

```
10  20  DESIGN  ' ~ '
```

```
 ~      ~      ~  ~ ~
~~ ~~~             ~ ~
  ~~   ~~~   ~   ~ ~
  ~          ~ ~~ ~~
 ~       ~ ~
 ~ ~~~ ~~~   ~~     ~
~ ~~  ~ ~     ~   ~~
  ~ ~     ~ ~        ~
    ~   ~      ~
~     ~~ ~~~
```

```
10  20  DESIGN  ' O  0  o  •  . '
```

```
o          OO     • o
OO    0  OO.• •    •o
0 0   0 0 oO• 0O   .
 •     0••0  0    ••0•
 •• oOOo0 o        0O
        •    0 .    0•o
.0•   o  0 OO .0 0
. 0       Oo 0          o
   oO.           0       o
.•   . . .         .   o.
```

```
10  20  DESIGN  '▯▯* *▯▯'
```

```
*   *▯▯ *▯▯▯▯▯*▯▯▯**▯
 ▯▯▯▯ ▯  ▯▯▯▯* **▯▯*
*▯▯▯▯▯ *▯▯▯*▯* ▯▯*▯▯
*▯ ▯▯▯▯▯*▯ ▯▯▯*▯▯
▯▯***▯▯▯**▯▯ ▯▯*▯▯▯▯
***▯**▯▯**▯▯* ▯▯*▯▯▯
* *****▯▯* **▯▯*▯▯*▯
*▯*▯▯*▯▯*▯*▯*▯  ▯*▯*
 *▯ *▯▯▯* ▯▯*▯ *▯▯▯▯
▯▯▯▯▯▯▯▯▯*▯* ▯▯*▯▯▯▯
```

While these "computer hieroglyphics" may have dubious aesthetic appeal, one can imagine -- instead of these typed symbols -- randomly generated swatches of color, perhaps displayed on a television-like screen.

Extensions of this approach to computer art include: (a) automating DESIGN, (b) weighting the selection of COLORS, (c) asking for human judgement (Do you like it or not?) in order to adjust weights on COLORS or other aesthetic factors, and (c) piecing together several computer-generated PICTUREs into a montage.

Another approach to computer art involves viewing programs which simulate artistic technique. For example, consider the program MONDRIAN below (named after the Dutch abstract painter).

```
      ∇ MONDRIAN
[1]   CANVAS←30 50⍴' '
[2]   DAB:COLOR←'o□*'[?3]
[3]   SIZE←3 5⌈?6 10
[4]   PICK:PLACE←?30 50-SIZE
[5]   OVERLAP←+/+/CANVAS[PLACE[1]+⍳SIZE[1];PLACE[2]+⍳SIZE[2]]≠' '
[6]   →PICK IF OVERLAP>2
[7]   CANVAS[PLACE[1]+⍳SIZE[1];PLACE[2]+⍳SIZE[2]]←COLOR
[8]   →DAB IF(PERCENT' 'ON CANVAS)>67
[9]   CANVAS
      ∇
```

MONDRIAN begins with a blank canvas (arbitrarily set at 30 by 50). Then the program chooses a random COLOR, SIZE and PLACE to DAB.

OVERLAP measures the extent of overlap with DABs already on the CANVAS.

IF OVERLAP is greater than 2, then it will PICK another PLACE. (This is tantamount to finding relatively open space on the CANVAS).

IF, however, OVERLAP is not too large, the COLOR is put on the CANVAS at the PLACE and in the SIZE selected.

The program continues to DAB IF the PERCENT of blank spaces ON the CANVAS is greater than 67. In other words, as soon as it is 1/3 filled up, CANVAS is displayed.

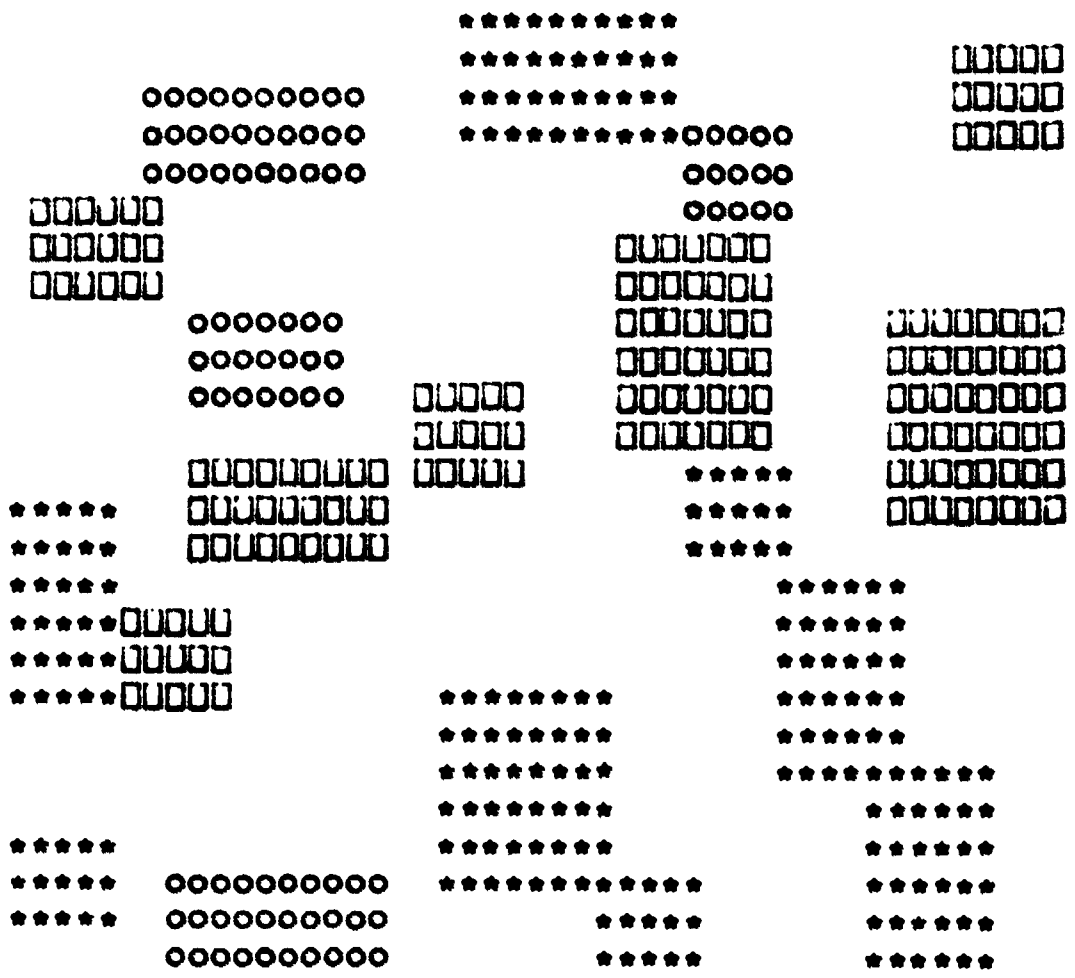
Note: MONDRIAN uses two simple sub-programs (mostly for readability).

They are PERCENT and ON:

```
      ∇ HUNDREDTHS←PERCENT N
[1]   HUNDREDTHS←⌊0.5+100×N
      ∇

      ∇ DENSITY←SYMBOL ON PICTURE
[1]   DENSITY←(+/+/SYMBOL=PICTURE)÷(×/⍴PICTURE)
      ∇
```

Now, MONDRIAN at work:



Possible extensions of this kind of program include: (a) simulating and combining additional artistic techniques (those that can be operationalized), (b) computing abstract measures of difference between random "paintings" produced by the computer, (c) converging to minimal differences from a previously specified "ideal" painting, and (d) developing a model for aesthetic judgment -- perhaps one which "evolves."

Conclusion

These are but a few APL "glass box" programs designed to stimulate students to think about selected concepts. Each of the sample programs shown here can be used as is and, of course, can be extended in a myriad of directions. Other topics well-suited for this pedagogical approach include some drawn from linguistics, statistics, mathematics, engineering, ecology, and physical sciences.

The challenge to educators, then, is to identify such topics suitable for embodiment as glass box programs, to search out the kernel concepts to be taught, and to lead students to better understandings of those concepts using a programming language.

References

[1] Papert, S. "Teaching Children Thinking", M.I.T. LOGO Memo #2, Oct. 1971.

[2] Iverson, K.E. "APL in Exposition", IBM Tech. Report #320-3010, Jan. 1972.

[3] Berry, P. et.al. "APL and Insight: The Use of Programs to Represent Concepts in Teaching", IBM Tech. Report #320-3020, March 1973.